COM00005I

BSc, BEng, MEng, MMath Examinations, 2012–2013

All Undergraduate Computer Science Programmes
Stage Two

Principles of Programming Languages (POPL)

Open Examination

Issued at:
Noon: Aut/8/Fri (30 November 2012)

Submission due:
Noon: Aut/10/Wed (12 December 2012)

Your attention is drawn to the Guidelines on Mutual Assistance and Collaboration in the Students' Handbook and the Department's 'Guide to Assessment Policies and Procedures' (http://www.cs.york.ac.uk/exams/statementonassessment/).

All queries on this assessment should be addressed to
Alan Wood (Part 1) `wood@cs.york.ac.uk`**, or**
Alan Burns (Part 2), `burns@cs.york.ac.uk`.

There are two parts in this assessment. You must complete both parts, which carry equal marks. Each part is divided into tasks, and the relative weight for each task is indicated as a percentage of the total marks for the assessment. All tasks must be attempted.

Submission

- All submissions should be *hard-copy* (printed), submitted to the Departmental reception by the hand-in deadline.
- Separate documents should be submitted for each part.
- Each item submitted should indicate clearly for which task it is the solution.
- Your examination number *must appear on each sheet* of your submission.
- Your name *must not appear* anywhere on your submission.

Part I.

There is one task in this part:

Task 1 Write an essay, of approximately 2500 words[1] (not including program code), with the following specification: 50%

Title:

A Comparison of Evaluation Strategies in Programming Languages

Abstract Expressions in programming languages are either 'primitive' or 'compound'. In the latter case, the sub-expressions can themselves be primitive or compound. It is therefore important to understand the ways in which the sub-expressions are evaluated. There are two main methods for sub-expressions evaluation: *normal-order* and *applicative-order*.

This essay will firstly explain these two strategies, comparing their similarities and differences. The uses of the two methods will be discussed, illustrating these with examples from actual programming languages. Finally, the essay will conclude with a critical summary of the pros and cons of applicative-order and normal-order evaluation.

Notes

1. Start your essay with the *exact* title and abstract as given above.
2. Use the principles of writing that were taught in the Stage 1 SKIL module.
3. Full references, properly formatted, to sources of material *must* be given.[2]
4. No more than half of the items in your bibliography can be to URLs alone — the others must be to 'printed' sources.
5. *Structure and Interpretation of Computer Programs*[1] gives a good introduction to the topic, so you might start from there. However, you *must* draw on other, fully-referenced, sources.

[1]About 5 typed (10pt), single-spaced, A4 pages
[2]Remember, we can Google as easily as you!

6. You *must* include appropriate program illustrations from at least *three* different programming languages, and these must all be syntactically correct. These illustrations should *not* be counted as part of the 2500 words of the essay.
7. In order to pitch the essay at the correct level, you should assume the reader has the technical knowledge of a typical CS student just starting Stage 2, who has not yet attended the POPL module.

Marking

Points for which marks will be given include:

- Clarity of explanation.
- Quality and choice of references.
- Appropriate program examples.
- Quality of argument.
- Clarity of essay structure.
- Quality of the conclusion.

Part II.

There are two tasks in this part:

$\boxed{\textbf{Task 2}}$ Write a short review, of approximately 500 words (at most one typed (10pt) A4 page), with the following specification: $\boxed{15\%}$

The Dining Philosophers Problem was first introduced by Dijkstra in 1965. Since then it has been used extensively to illustrate various features of concurrent languages and programs.

Using sources available on the internet (in addition to Wikipedia) consider how this problem has been discussed and solved since 1965. You do not need to describe the problem, or give code from solutions. But you should consider which language features and which concurrent programming languages have used the problem as an illustration.

$\boxed{\textbf{Task 3}}$ A different variation of the Dining Philosophers Problem also has a restricted number of plates. Each Philosopher needs two forks and a plate in order to eat. The forks are positioned between the Philosophers (as in the original problem). For N Philosophers there are N-2 plates (placed at the centre of the table).

Develop a program that implements this system for *five* Philosophers.

Your program should represent each Philosopher as a task/thread and use monitors to control access to the resources (i.e. the forks and plates). To simulate the act of eating and thinking a delay/sleep routine should be used.

You can program this is in Ada, Java or Pascal-FC. A description of your code should also be provided, including any further assumptions you make about the problem. Example runs of your program should be provided.

You should indicate what steps you have taken to prevent deadlocks (or livelocks) in your program.

Note the program must implement the above variant of the Dining Philosophers Problem, not the original problem. Only one language solution is required. If more than one program is submitted only the first program will be marked. $\boxed{35\%}$

References

[1] H. Abelson, G.J. Sussman, and J. Sussman. *Structure and Interpretation of Computer Programs*. MIT Press, 1985.